# BG/L System Overview

Susan Coghlan

## MCS BG/L Specs

- Core Rack:
  - 1024 dual-cpu 440 700Mhz 512MB RAM compute nodes
  - 32 I/O nodes (1/32 computes, basically the same as compute nodes),
  - Torus, Global Tree, Global Interrupt networks
- Frontends:
  - JS20 blades – PPC970 2.1GHz dual-cpu 4GB RAM
- Service Node:
  - 4-way 1.7 Ghz PPC (2 CPU cores), 16GB RAM
- 20 Storage servers (4 homedir, 16 PVFS) ~14TB

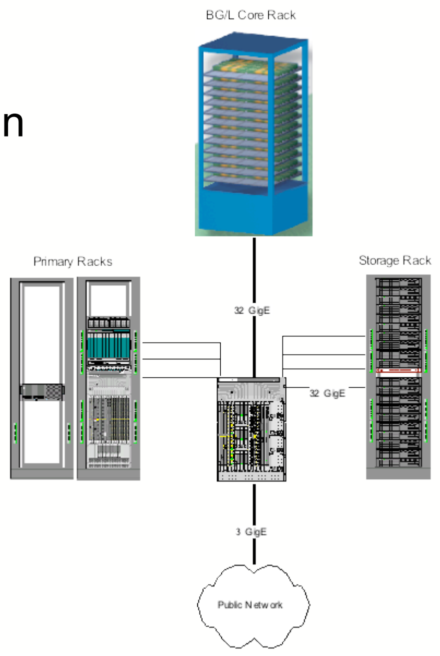
## Core Rack Layout

- 2 Midplanes per rack (top and bottom)
- 16 Node Boards per Midplane
- 16 Compute Cards per Node Board
- 2 Chips per Compute Card
- 2 Processors per Chip
- 1 I/O assigned to each set of 32 Compute Nodes

## Layout of BGL

- ***Login servers*** – SuSE Linux OS (SLES9), regular user logins, compiles and job submissions
- ***Service node*** – SuSE Linux OS (SLES8), no regular user logins, job manager, partitioning of cluster, boots I/O nodes
- ***I/O nodes*** - embedded Linux, no logins yet, function is to forward I/O between compute nodes and storage nodes
- **Storage/home nodes** – SuSE Linux OS (SLES9), filesystems servers, no regular users logins, provides filesystems for job access, home dirs on login servers
- **Compute nodes** – IBM microkernel, no shell, no dynamic library loading, no local filesystems

## MCS BG/L Configuration



## Accessing BGL

- Request BGL resource from MCS account pages https://accounts.mcs.anl.gov
- Welcome mail with information on how to log on, running a simple job, etc. will be sent to you
- SSH keys might already be set up for you, if not, send your public key to support@bgl.mcs.anl.gov
- SSH into **bgl.mcs.anl.gov** from an ANL machine, otherwise, ssh into MCS conduit machine (terra) – forwarding your ssh keys through to BGL

## Job Run Basics

- Overview:
  - Partitions are **allocated (or booted)**
    - **Partition is set for a specific user**
    - **I/O nodes associated with partition boot with specified ramdisk and kernel**
    - **I/O nodes NFS mount /bgl, home and eventually PVFS directories**
  - **User job(s) is (are) run**
  - **Partition is freed**
- Compute Node run modes:
  - Co-Processor (default) - 1 proc computes, 1 handles communication
  - Virtual – both procs are used for computing

## Partitions explained

- Partitions consist of I/O node(s) and compute nodes
- 3 standard partitions: full (1024), midplane0 (512), midplane1 (512)
- Special 32 node partitions: Pgeneral for any user's use, dedicated P<username> for specific users.  Ask if you need a dedicated partition
- 2 modes of using partitions:
  - Pre-allocated
  - Allocated on the fly
- Allocating a partition takes around 3 minutes
- Multiple jobs (by the same user) can be run within a pre-allocated partition

## mpirun

- Example mpirun command:
  mpirun -partition Psmc -np 32 -cwd `pwd` -exe `pwd`/hello.rts
- To run jobs on a specific partition use mpirun -partition <partition> (always specify a partition)
- -np <#procs> can be <= partitionsize for C mode, <= 2*partitionsize for V mode (partition must be set to V mode default to use V mode)
- mpirun always used to run jobs – whether they are mpi based or not
- A job already running in a partition gives a reasonable error. It causes no damage to try to run.

## Scheduling Jobs

- No job scheduler at this time. Narayan Desai is writing one. We hope to have a version available next week.
- Reservations are needed for anything larger than a 32 node partition.
- Reservations are handled by notifying the users and asking them not to run jobs during a specific time period.
- We are working on tools to list jobs, kill jobs, etc.
- If you have a hung job that you can't kill, contact support.

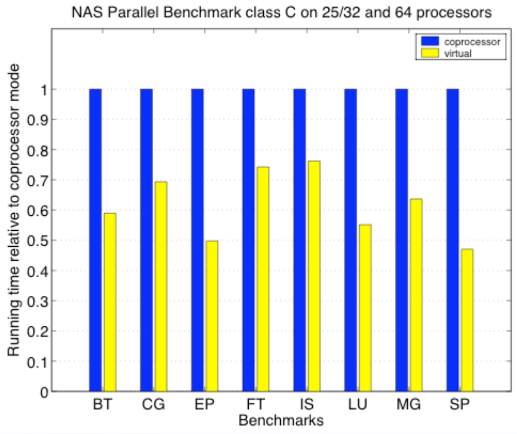
## Help

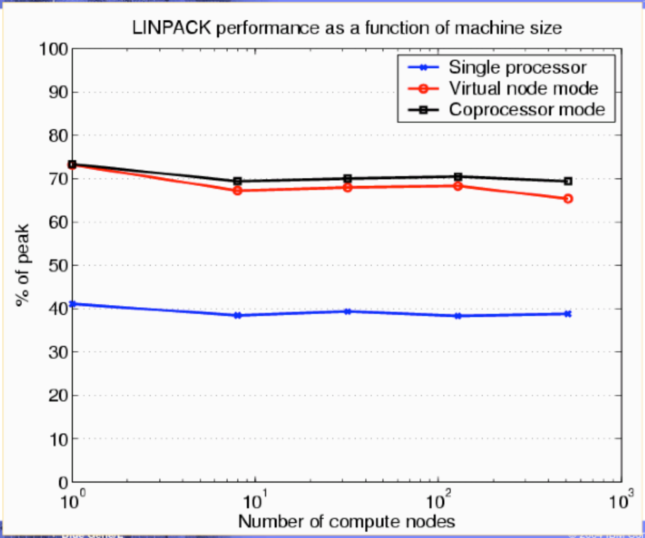
- Documentation on the web site: http://www.bgl.mcs.anl.gov
- Mailing list archives – discuss, notify: http://www.bgl.mcs.anl.gov/MailLists
- Email to discuss@bgl.mcs.anl.gov for user community support
- Email to support@bgl.mcs.anl.gov for system support
- Web problem tracking report (not yet available) http://www.bgl.mcs.anl.gov/support

## Parallel Issues on BG/L

Katherine M Riley

## NAS


NAS Parallel Benchmark class C on 25/32 and 64 processors
coprocessor
virtual
Running time relative to coprocessor mode
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
BT
CG
EP
FT
IS
LU
MG
SP
Benchmarks


LINPACK performance as a function of machine size
Single processor
Virtual node mode
Coprocessor mode
% of peak
0
10
20
30
40
50
60
70
80
90
100
$10^0$
$10^1$
$10^2$
$10^3$
Number of compute nodes

## The Networks

- 3D Torus
  - Point-to-Point, Point-to-Some
- Global Tree
  - Global Operations, 8 microseconds latency
- Global Barriers and Interrupts
  - Low latency barriers and interrupts (over a tree)
- G-bit Ethernet
  - File I/O and Host interface
- Control Network
  - Boot, Monitoring and Diagnostics

## Torus

- Each node has 6 nearest neighbor interconnects
- Raw hardware bandwidth:
  - 2bits/cycle -> 175MB/s @ 700 MHz
  - 6 microseconds
- Adaptive routing
- Maximum packet size of 256 bytes
- Turn it off :
  - "`mpirun -connect MESH`"

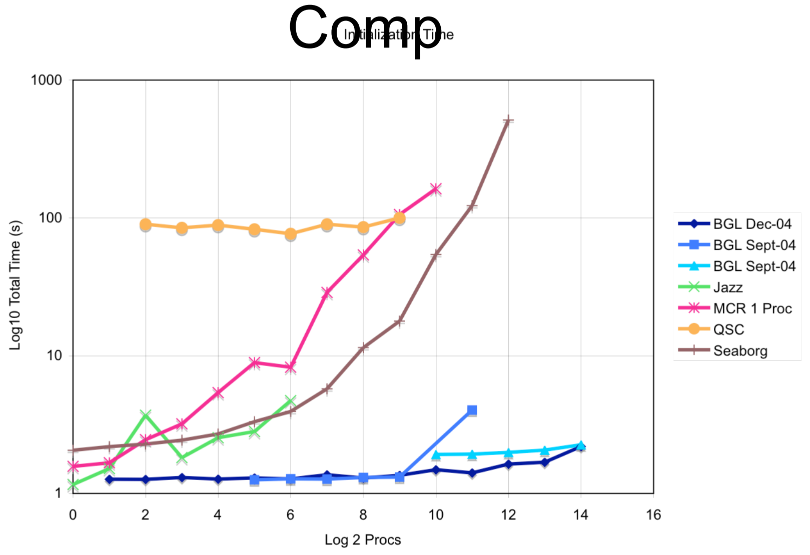
## Noise

- Very little on the system
  - Tiny kernel doing very little
    - Results in limited node functionality
- Necessary for coordinating communication
  - It really works
- Very reproducable results
- Can use almost the entire 512MB

## Using the Second CPU

- Communication Coprocessor Mode
  - “`mpirun -mode CO`”
- Virtual Node Mode
  - “`mpirun -mode VN`”
  - Two MPI processes per node (1 per proc)
    - Communication through L3 cache
  - Issues
    - Share the 512 MB
    - No communication offload

## Massive Communication + Comp



## Controlling MPI

- Setting parameter like eager
  - -env BGLMPI_EAGER=1000
- Control networks for individual commands
  - BGLMPI_ALLREDUCE=MPICH,TREE, TORUS
    - Available for all global operations
  - BGLMPI_BARRIER

## Message Passing for Applications

- Communicators
  - Tree interconnect utilized only for MPI_Comm_world operations
  - Torus interconnect supports multicast operations on any axis
    - Special topology-aware algorithm to fill plane
- Map MPI apps to machine topology

## Create Physical Node Partition

- User specified shape of partition when submitting job
  - `mpirun -shape NxNxN`
- Contiguous rectangular partition
- Nodes indexed (x,y,z) coordinates inside job partition

## Partition Mapping

- Can physically map processes to processors
  - Create mapping file
  - More info to follow
- Also - control how processes are filled
  - BGLMPI_MAPPING=TXYZ
    - In VN, fill both procs first, then nodes
  - BGLMPI_MAPPING=XYZT
    - Default - fill nodes then procs

## Profiling

- Visualizers
  - Efforts for jumpshot and paraver
- Simpler Characteristics
  - FPMPI
  - FPMPI-like from IBM
- Compiler provides access to routine wrapping
  - `"-finstrument-functions”`
  - `__cyg_profile_func_enter(void * this_function, void * call_site)`
  - `__cyg_profile_func_exit(void * this_function, void * call_site)`

## Debugging

- Gdb
  - Work in progress, trying to get it
  - Connect to hung processes
- Totalview
  - Ported, but, unlikely to come here

## Aspects of single processor optimization on BG/L

Andrew Siegel

## General CPU info

- Two processors per node
  - No support for SIMD between processors
  - Very unsafe dual-proc programming model (co_start(…)) that is not recommended for apps programmers
  - *Virtual node* or *co-processor* mode recommended
- 32-bit architecture at 700MHz
- Single integer unit
- Single load/store unit
- Special double fpu (double hummer)

## General, cont.

- L1 data cache: 32Kb, 32-Byte line size
- L2 data cache: Prefetch buffer: 16 128-byte lines
- L3 data cache: 4Mb, 35 cycle latency
- Memory: 512Mb DDR at 350MHz, ~85 cycles latency
- Double FPU: 32 primary (P) and 32 secondary (S) fp registers -- major key to single CPU performance:
  - Standard PowerPC instructions on fpu0 (fadd, fmadd, fadds, fdiv)
  - Typical SIMD instructions for 64-bit floating point (fpadd, fpmadd, …) for P and S
  - Also SIMOMD: cross, asymmetric, and complex operations
  - Quad-word load/store (pair of 64-bit fp's one to each group of registers)

## SIMOMD FMA Instructions

| **FMA instruction class** | **Example** | **Operation** | **C99 builtins** |
|---|---|---|---|
| Parallel | fxmadd fT, fA,fC,fB | $P_T = P_A*P_C+P_B$ <br> $S_T = S_A*S_C+S_B$ | T=_fpmadd(B,C,A) |
| Cross | fxmadd fT,fA,fC,fB | $P_T = P_A*S_C+P_B$ <br> $S_T = S_A*P_C+S_B$ | T=_fxmadd(B,C,A) |
| Replicated | fxcpmadd fT,fA,fC,fB | $P_T = P_A*P_C+P_B$ <br> $S_T = P_A*S_C+S_B$ | T=_fxcpmadd(B,C,$a_p$) |
| Asymmetric | fxcpnpma fT,fA,fC,fB | $P_T = -P_A*P_C+P_B$ <br> $S_T = P_A*S_C+S_B$ | T=_fxcpnpma(B,C,$a_p$) |
| Complex | fxcxnpma fT,fA,fC,fB | $P_T = -S_A*S_C+P_B$ <br> $S_T = S_A*P_C+S_B$ | T=_fxcxnpma(B,C,$a_s$) |

## Instruction throughput

| Instruction | Latency (cycles) | Throughput/cycle |
|---|---|---|
| fadd | 5 | 1 |
| fmadd | 5 | 1 |
| fpmadd | 5 | 1 |
| fpdiv | 30 | 1/30 |

Theoretical limit: 1fpmadd/cycle = 4 FLOPs/cylcle = 2.8GFLOPs

## Random notes

- Warning: no hardware sqrt function
  - GNU implementation ~100 cycles
  - Others up to 5x faster
  - Other math libs still not fully optimized
- Efficient use of double-hummer requires 16-byte alignment for quad-word load/store instructions
- lfpd/stfpd can double the bandwidth to L1-cache

## Data alignment

- Compiler will align data greater than 16 bytes to 16-byte boundaries for <u>stack locals</u> and <u>externally defined</u> data. Special 16-byte-aligned malloc now also available.
- Parallel load/store is **suppressed** unless it can be determined by examination of alignment information on memory references that instruction will not trap
- Often, it is not possible for compiler to tell if a pointer point to aligned data. (this is currently being improved)
- User can assert so using:
  - _alignx(16, f);  C
  - call alignx(16, f(1))  Fortran

## Data alignment sample code

**Fortran :**

```
  call alignx(16,x(1))
  call alignx(16,y(1))
!ibm* unroll(10)
  do i = 1, n
    y(i) = a*x(i) + y(i)
  end do
```

**C :**

```
double * x, * y;
#pragma disjoint (*x, *y)
__alignx(16,x);
__alignx(16,y);
#pragma unroll(10)
for (i=0; i<n; i++) y[i] = a*x[i] + y[i];
```

Try : -O3 -qarch=440d -qlist –qsource

## Compiler Use

- Optimization levels:
  - Default optimization = none (very slow)
  - -O : good place to start, use with -qmaxmem=64000
  - -O2: same as -O
  - -O3 -qstrict : less aggressive, must strictly obey program semantics
  - -O3: aggressive, allows re-association, will replace division by multiplication with the inverse
  - -qhot : turns on high-order transformation module, will add vector routines, unless -qhot=novector
  - check listing: -qreport=hotlist
  - -qipa : inter-procedure analysis; many suboptions such as: -qipa=level=2
- Architecture flags:
  - -qarch=440 : generates standard powerpc floating-point code
  - -qarch=440d : will try to generate double FPU code

## Compiler, cont.

- Recommendation:
  - On BG/L start with : -g -O -qarch=440 -qmaxmem=64000
  - Try : -O3 -qarch=440/440d
  - Try : -O5 -qarch=440d
  - -O4 = -O3 -qhot -qipa=level=1 -qarch=auto
  - -O5 = -O3 -qhot –qipa=level=2 -qarch=auto
- Warning: Generated .s files not valid! Must use –qreport -qsource to get assembler listing and double hummer info!

---

- There are several compiler components. The backend code generator has some vectorization capability, in fact it is very good in certain cases.
- You would get that with -O3 -qarch=440d. The -qtune=440 option is set by default in the compiler config file, so you don't need it unless you use -O4 or -O5. The higher optimization levels are:

  -O4 = -O3 -qhot -qipa -qarch=auto -qtune=auto
  -O5 = -O3 -qhot -qipa=level=2 -qarch=auto -qtune=auto

  The -qhot option invokes a different compiler module with more vectorization capability.
- Lore to date: often get better results from -O3 -qarch=440d, but the -qhot capability should improve over time, and it can do transformations that the backend peice simply can't handle.

## Profiling

- BG/L is similar to most risc systems, so profiling has been done to this point on another platform.
- Standard profiling (prof, gprof) is not yet available on BG/L so use IBM Power4 systems for profiling.
- Major differences between BG/L and IBM Power4:
  - BG/L has minimal L2 => BG/L benefits more from L1 cache re-use
  - BG/L does not have a hardware square root
  - Mathematical intrinsic routines (exp, log, …) on BG/L are from GNU libm.a =>
- Good approach: profile on IBM Power4 / AIX using xprofiler
- Use MPI profiling tools – there are many to choose from.
  - mpi_trace : low overhead, text summary for free (see demo later)
  - paraver : extensive data analysis capabilities
  - mpe / jumpshot : standard with MPICH

## Math Routines

- Scalar and Vector MASS Routines
- Approximate cycle-counts per evaluation on BG/L

| op | libm | libmass | libmassv |
|---|---|---|---|
| **exp** | 185 | 64 | 22 |
| **log** | 320 | 80 | 25 |
| **pow** | 460 | 176 | 29-48 |
| **sqrt** | 106 | 46 | 8-10 |
| **rsprt** | 136 | … | 6-7 |
| **1/x** | 30 | … | 4-5 |

## Floating Point Counters:

- 2 counters per core
- Can count 1 set  of load/store and 1 set of arithmetic ops concurrently.
- Load / Store Sets:
  - One of {Double LD, Double ST, Quad LD, Quad ST}
- Arith Op Sets:
  - One of {Adds, Mults, FMA, All Quad Arithmetic Ops}
- Need multiple runs to get all possible sets.

## Bgl_perfctr

- API abstraction of the hardware performance counter
- 64bit virtualization of 32b counters
- Overflow protection using timer interrupts
- Mnemonic names for all events
- Unified view of all events without limitations of functionality
- Transparent event to counter mapping

## PAPI

- De Facto standard API for user level tools and application developers
- BG/L implements PAPI v. 2.3.4
- Full support for create, remove, start, stop, reset event sets
- All BG/L counters are supported in PAPI using the native event format

```
(bgl_event_mnemonic & 0x3FF) | (bgl_edge)<<10
```

## New PAPI events

- PAPI_BGL_OED
  Oedipus instructions in FPU unit
- PAPI_BGL_TS_32B
  Total number of 32B chunks sent in any torus direction
- PAPI_BGL_TR_DPKT
  Total number of tree data packets sent on any channel
- PAPI_BGL_TS_FULL, PAPI_BGL_TR_FULL
  Cycle counts indicative of pile-up

## Optional PAPI functionality

- PAPI can support some functions that is available on some computing platforms.
- On platforms where the functionality is not available these API calls return an error code
- On BG/L this includes the following
  - PAPI_overflow(): To call a user handler on counter reaching a threshold value
  - PAPI_multiplex(): Automatic, periodic switching between event sets. Use different counter set up on different nodes instead.